



Intro to Java

Data Types, Console I/O,
Conditional Statements, Loops,
Bitwise Operations



SoftUni Team
Technical Trainers
Software University
<http://softuni.bg>

Advanced
Java



```
public class HelloJava {  
    public static void main(String[] args) {  
        System.out.println("Hello Java!");  
    }  
}
```

Table of Contents

1. Console-Based Input and Output
2. Data Types
3. Conditional Statements
4. Loops
5. Bitwise operations



Have a Question?

sli.do

#JavaAdvanced



Console Input and Output

Scanner and Formatted Printing

Reading From the Console

- The **java.util.Scanner** class can read **strings**

```
import java.util.Scanner;  
...  
  
Scanner scanner = new Scanner(System.in);  
String name = scanner.nextLine();
```

- The **scanner** takes as a parameter an input stream
- **nextLine()** reads a whole line
- **next()** reads the string before the next delimiter

Reading From the Console (2)

- The **java.util.Scanner** class can read **numbers**

```
import java.util.Scanner;  
...  
  
Scanner scanner = new Scanner(System.in);  
int firstNum = scanner.nextInt();  
double secondNum = scanner.nextDouble();
```

- The numbers **can be separated** by any sequence of whitespace characters (e.g. spaces, tabs, new lines, ...)
- **Exception is thrown** when non-number characters are entered

Printing to the Console

- Using **System.out.print()** and **System.out.println()**:

```
String name = "SoftUni";  
String location = "Sofia";  
double age = 3.5;
```

Prints **without** a
new line

```
System.out.print(name);  
System.out.println(" is " + age +  
    " years old organization located in " + location + ".");
```

Prints a new line

```
// Output:  
// SoftUni is 3.5 years old organization located in Sofia.
```

Problem: Read Input

- Write program that reads:
 - **Two words** from the first line
 - **Integer and two doubles** which may be on multiple lines
 - A **string** from the next line
- Prints **{firstWord} {secondWord} {thirdWord} {(int) sum}**

```
Java Rocks  
5      12.5    -7.5  
End
```



```
Java Rocks End 10
```

Check your solution here: <https://judge.softuni.bg/Contests/382>

Solution: Read Input

```
Scanner scanner = new Scanner(System.in);

String firstWord = scanner.next("\\w+");
String secondWord = scanner.next("\\w+");
int firstInt = scanner.nextInt();
double firstDouble = scanner.nextDouble();
double secondDouble = scanner.nextDouble();
scanner.nextLine(); // Skip to the line end
String thirdWord = scanner.nextLine();

int sum = firstInt + firstDouble + firstDouble;

System.out.println(firstWord + " " + secondWord + " " + thirdWord + " " + sum);
```

Check your solution here: <https://judge.softuni.bg/Contests/382>

Formatted Printing

- Java supports **formatted printing** by **`System.out.printf()`**

- `%s`** – prints a string argument

```
System.out.printf("Name: %s", name);
```

- `%f`** – prints a floating-point argument

```
System.out.printf("Height: %f", height);
```

- `%.2f`** – prints a floating-point argument with 2 digits precision

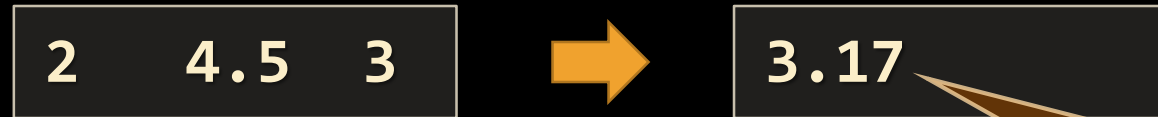
```
System.out.printf("Height: %.2f", height);
```

- `%n`** – prints a new line

- Learn more at <http://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html>

Problem: Average of Three Numbers

- Write program that reads **three** numbers.
- Print the **average** of the three



```
Scanner scanner = new Scanner(System.in);  
double first = scanner.nextDouble();  
double second = scanner.nextDouble();  
double third = scanner.nextDouble();
```

```
double sumAbs = first + second + third;  
double avg = sumAbs / 3;
```

```
System.out.printf("%.2f", avg);
```

Round up to 2nd digit

Check your solution here: <https://judge.softuni.bg/Contests/382>



Data Types in Java

Integer, Double, String, Boolean

Floating-Point Types

- Floating-point types are:
 - **float** ($\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$)
 - 32-bits, precision of 7 digits
 - **double** ($\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$)
 - 64-bits, precision of 15-16 digits
- The default value of floating-point types:
 - Is **0.0F** for the **float** type
 - Is **0.0D** for the **double** type

double

float



3.141592653589793238462643383
279502884197169399375105820974944
59320781640628620899862803482534211
70679821480865132822066479938446085
50982251775359408 124481117
45028410 270193852 116555944
622948 954930381 964428809
75 668933446 1284756482
3378678116 527120109
145648546 9284603486
104543264 821339307
2602491812 7372458700
66033155881 74881520920 962829
2509171536 43678925003600113805
305488204652 1384146931944911609
43305727036875 958195309218611738
19326117931051 18548074462379962
7495673518657 527248912270381
8301494912 9833673362
44065 66430

Integer Types

- **byte** (-128 to 127): signed 8-bit
- **short** (-32,768 to 32,767): signed 16-bit
- **int** (-2,147,483,648 to 2,147,483,647): signed 32-bit
- **long** (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807): signed 64-bit

```
byte b = 1;  
int i = 5;  
long num = 3L;  
short sum = (short) (b + i + num);
```

Capital I is harder to
confuse with 1

int **short**
long **byte**

Floating-Point Types – Examples

```
float f = 0.33f;
```

"f" specifies a float

```
double d = 1.67;
```

```
double sum = f + d;
```

Double by default

```
float fSum = f + d; // This will not compile
```

```
double infinity = 3.14 / 0;
```

```
System.out.println(f); // 0.33
```

```
System.out.println(d); // 1.67
```

```
System.out.println(sum); // 2.0000000013113022
```

```
System.out.printf("%.2f", sum); // 2.00
```

```
System.out.println(infinity); // Infinity
```

- The floating-point arithmetic sometime works incorrectly
 - Don't use **float** and **double** for financial calculations!
- In Java use the **BigDecimal** class for financial calculations:

```
import java.math.BigDecimal;  
...  
BigDecimal bigF = new BigDecimal("0.33");  
BigDecimal bigD = new BigDecimal("1.67");  
BigDecimal bigSum = bigF.add(bigD);  
System.out.println(bigSum); // 2.00
```


Problem: Euro Trip

- Write program that reads:
 - **Quantity of a product** from the first line
- **Prints the amount** of Deutsche Marks needed to buy it
- Exchange rate: 4210500000000 : 1, price of 1kg product : 1.20 BGN

2.35	→	11873610000000.00 marks
1		5052600000000.00 marks
15		75789000000000.00 marks

Check your solution here: <https://judge.softuni.bg/Contests/382>

Solution: Euro Trip

```
Scanner scanner = new Scanner(System.in);

double quantity = Double.parseDouble(scanner.nextLine());

double pricePerKilo = 1.20;
BigDecimal priceInLevs = new BigDecimal(pricePerKilo * quantity);

BigDecimal exchangeRate = new BigDecimal("4210500000000");
BigDecimal marksNeeded = exchangeRate.multiply(priceInLevs);

System.out.printf("%.2f marks", marksNeeded);
```

Check your solution here: <https://judge.softuni.bg/Contests/382>

Other Primitive Data Types

■ Boolean

```
boolean b = true;  
System.out.println(b); // true  
System.out.println(!b); // false
```

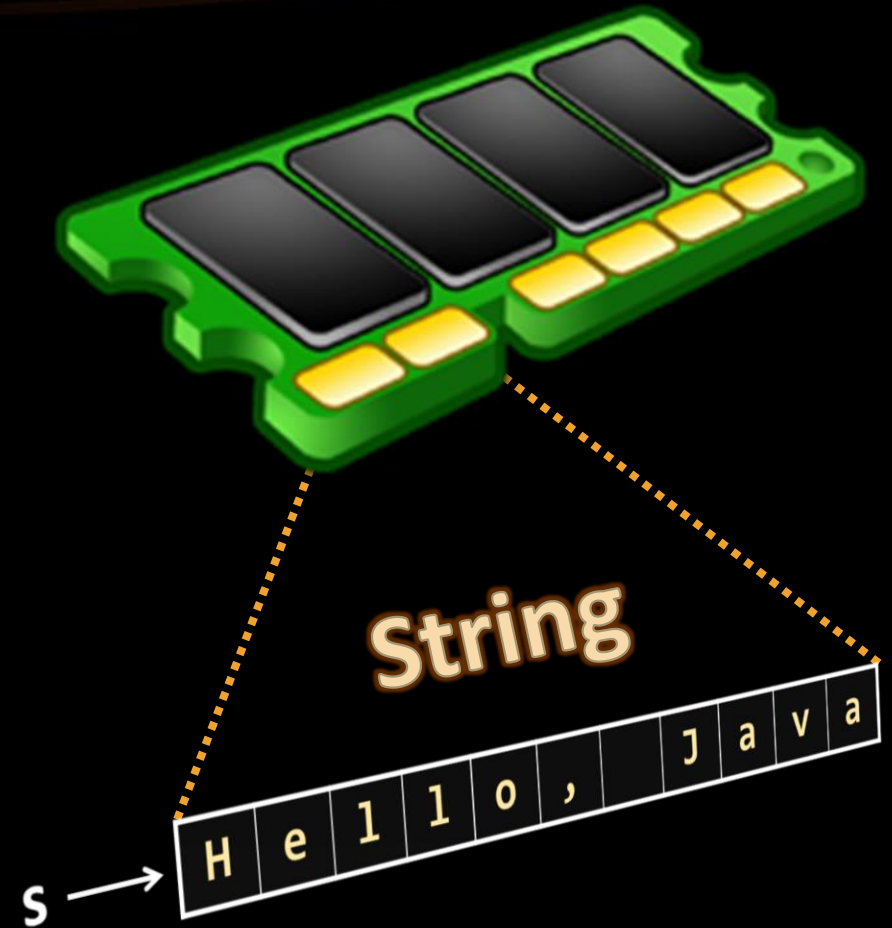
■ Character

```
char ch = 'ю';  
System.out.println(ch);  
char ch = '\u03A9'; // Ω  
System.out.println(ch);
```

The String Data Type

- The **String** data type:
 - Declared by the **String** class
 - Represents a sequence of characters
 - Has a default value **null** (no value)
- Strings are enclosed in quotes:

```
String s = "Hello, Java";
```
- Strings can be concatenated
 - Using the **+** operator



Problem: Greeting

- Read first and last name as an input.
- Print a greeting: "Hello, " + {firstName} {lastName} + "!"
 - if a **name is missing**, replace it with **five stars** '*'



Check your solution here: <https://judge.softuni.bg/Contests/382>

Solution: Greeting

```
Scanner scanner = new Scanner(System.in);

String firstName = scanner.nextLine();
String lastName = scanner.nextLine();

if (firstName.isEmpty()) {
    firstName = "*****";
}

// TODO: check last name

System.out.printf("Hello, %s %s!", firstName, lastName);
```

Check your solution here: <https://judge.softuni.bg/Contests/382>

The Object Type

- The **Object** type:
 - Is declared by the **java.lang.Object** class
 - Is the base type of all other types
 - Can hold values of any type

Object



```
object dataContainer = 5;  
System.out.print("The value of the dataContainer is: ");  
System.out.println(dataContainer);  
  
dataContainer = "Five";  
System.out.print("The value of the dataContainer is: ");  
System.out.println(dataContainer);
```



Nullable Types: Integer, Long, Boolean, ...

- Each primitive type in Java has a corresponding **wrapper**:
 - `int` → `java.lang.Integer`
 - `double` → `java.lang.Double`
 - `boolean` → `java.lang.Boolean`
- Primitive wrappers can have a value or be **null** (no value)

```
Integer i = 5; // Integer value: 5  
i = i + 1; // Integer value: 6  
i = null; // No value  
i = i + 1; // NullPointerException
```


Type Conversion

- Type **conversion** and **typecasting** change one type to another
 - Java supports **explicit** type conversion (casting)

```
float heightInMeters = 1.74f; // Explicit conversion  
double minHeight = (double) heightInMeters; // Explicit  
byte dataLoss = (byte) 12345; // Explicit with data loss
```

- Java also supports **implicit** type conversion

```
double maxHeight = heightInMeters; // Implicit  
double minHeight = heightInMeters; // Compilation error!
```

Implicit casting is not allowed
if data loss is possible

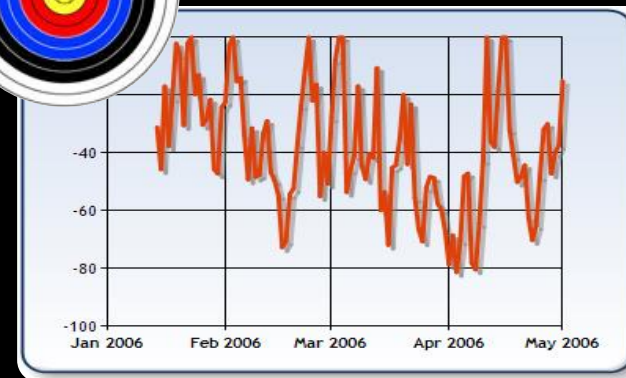
double



float



long



int

Practice: I/O and Data Types

Live Exercises in Class (Lab)

IF THEN ELSE

Conditional Statements
Implementing Conditional Logic

Conditional Statements: if-else

- Java implements the classical **if** / **if-else** statements:

```
Scanner scanner = new Scanner(System.in);
int number = Integer.parseInt(scanner.nextLine());

if (number % 2 == 0) {
    System.out.println("This number is even.");
} else {
    System.out.println("This number is odd.");
}
```


Conditional Statements: switch-case

- Java implements the classical **switch-case** statements:

```
switch (day) {  
    case 1: System.out.println("Monday"); break;  
    case 2: System.out.println("Tuesday"); break;  
    case 3: System.out.println("Wednesday"); break;  
    case 4: System.out.println("Thursday"); break;  
    case 5: System.out.println("Friday"); break;  
    case 6: System.out.println("Saturday"); break;  
    case 7: System.out.println("Sunday"); break;  
    default: System.out.println("Invalid day!"); break;  
}
```

Problem: Transport Price

- A student travels **n** kilometers using one type of transport:
 - **Taxi**: Initial tax: 0.70 USD. **Daytime** cost: **0.79** USD/km.
Night-time cost: **0.90** USD/km.
 - **Bus**: **Day / Night** tariff: **0.09** USD/km for at least **20 km**.
 - **Train**: **Day / Night** tariff: **0.06** USD/km for at least **100 km**.



Check your solution here: <https://judge.softuni.bg/Contests/382>

Solution: Transport Price

```
Scanner scanner = new Scanner(System.in);
int distance = Integer.valueOf(scanner.nextLine());
String dayOrNight = scanner.nextLine();

double taxiRate = 0.90;
if (dayOrNight.equals("day")
    taxiRate = 0.79;

if (distance < 20)
    System.out.printf("Taxi: %f", 0.70 + (distance * taxiRate));
else if (distance < 100)
    System.out.printf("Bus: %f", distance * 0.09);
else
    System.out.printf("Train: %f", distance * 0.06);
```

Check your solution here: <https://judge.softuni.bg/Contests/382>



Loops

Implementing Cyclic Structures

While Loop

- The simplest and most frequently used loop

```
while (condition) {  
    statements;  
}
```



- The repeat **condition**
 - Returns a boolean result of **true** or **false**
 - Also called **loop condition**

Problem: Numbers 0...9

- Using a while loop, print the numbers from 0 to 9 inclusive:

```
int counter = 0;
while (counter < 10) {
    System.out.printf("Number : %d\n", counter);
    counter++;
}
```

```
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
```

Check your solution here: <https://judge.softuni.bg/Contests/382>

Do-While Loop

- Another classical loop structure is:

```
do {  
    statements;  
}  
while (condition);
```



- The block of **statements** is repeated
 - While the boolean loop **condition** is true
- The loop is executed at least once

Problem: Product of Numbers [N..M]

- Calculate the product of all numbers in the interval [**n**..**m**]:

```
int n = scanner.nextInt();
int m = scanner.nextInt();
BigInteger product = BigInteger.ONE;

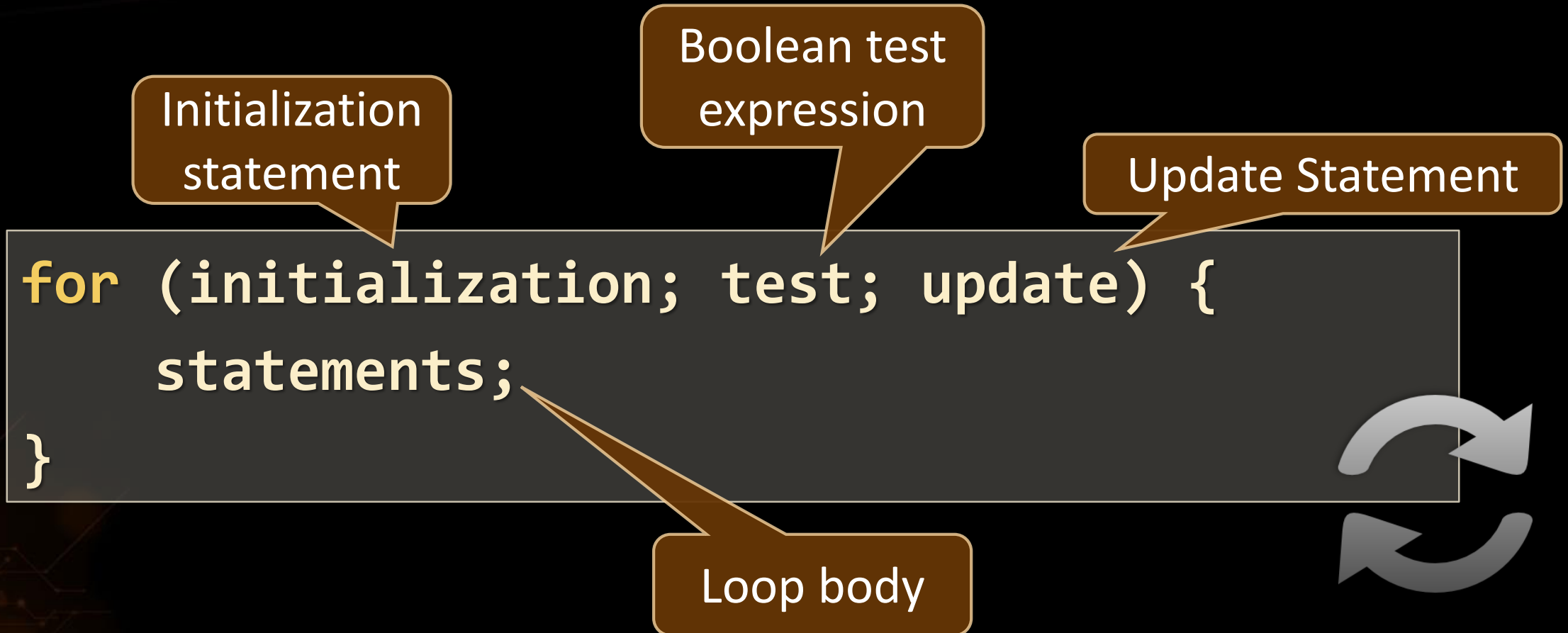
do {
    BigInteger number = new BigInteger("" + n);
    product = product.multiply(number);
    n++;
} while (n <= m);

System.out.printf("product[%d..%d] = %d\n", n, m, product);
```

Check your solution here: <https://judge.softuni.bg/Contests/382>

For Loops


- The classical **for**-loop syntax is:



Using the `continue` Operator

- **`continue`** bypasses the iteration of the inner-most loop
- Example: sum all odd numbers in $[1..n]$, not divisors of 7:

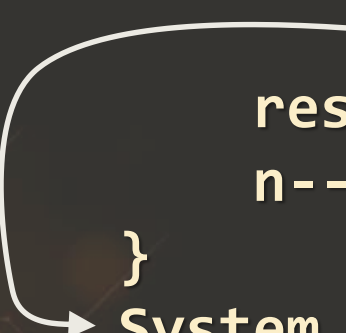
```
int n = 100;
int sum = 0;
for (int i = 1; i <= n; i += 2) {
    if (i % 7 == 0) {
        continue;
    }
    sum += i;
}
System.out.println("sum = " + sum);
```



Using the break Operator

- The **break** operator exits the inner-most loop

```
public static void main(String[] args) {  
    int n = new Scanner(System.in).nextInt();  
    // Calculate n! = 1 * 2 * ... * n  
    int result = 1;  
    while (true) {  
        if (n == 1)  
            break;  
        result *= n;  
        n--;  
    }  
    System.out.println("n! = " + result);  
}
```

A white curved arrow originates from the **break;** statement inside the while loop and points to the closing curly brace of the while loop, illustrating that the break operator exits the inner-most loop.

For-Each Loop

- The typical **for-each** loop syntax is:

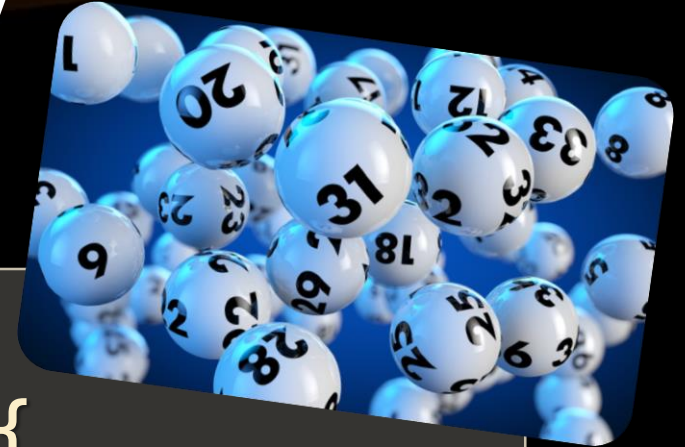
```
for (Type element : collection) {  
    statements;  
}
```



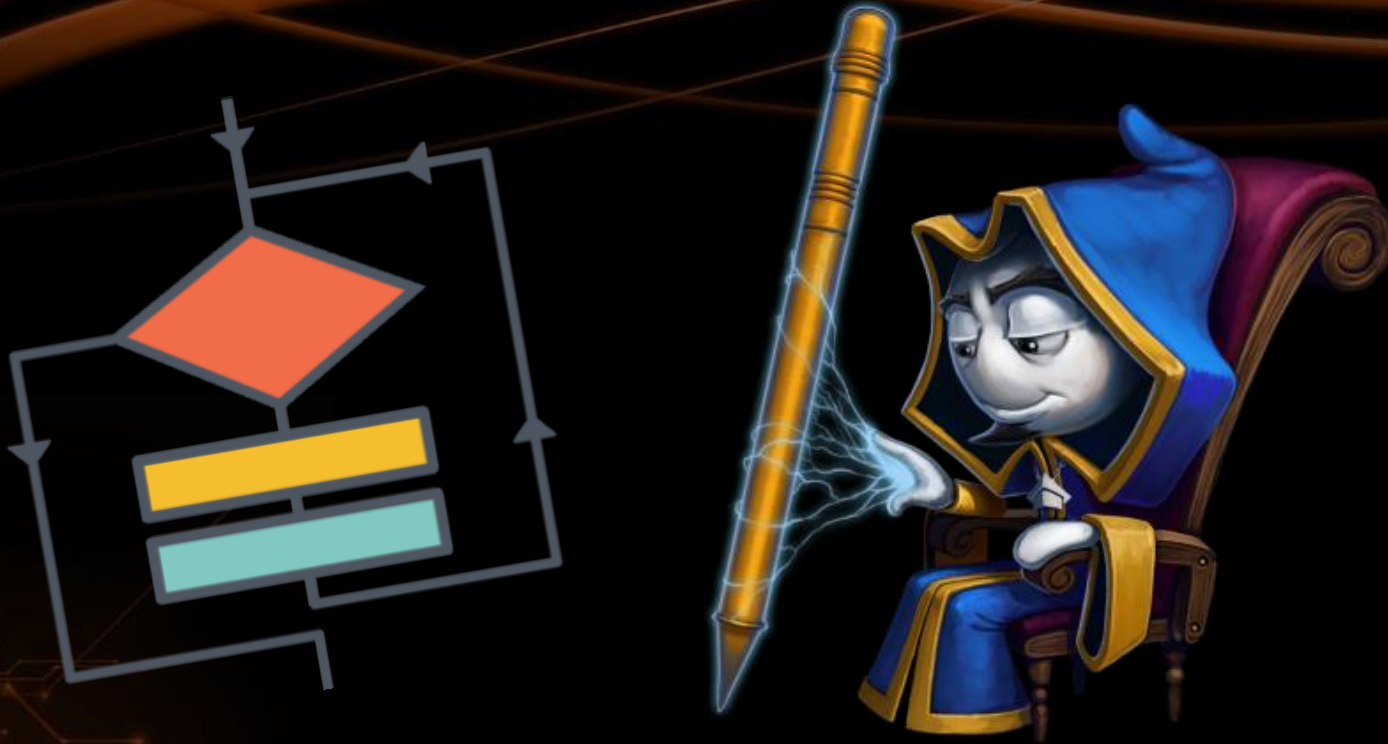
- Iterates over all the elements of a collection
 - The **element** takes sequentially **all** collection values
 - The **collection** can be any group of elements of the same type

Problem: Lottery

- **Print all combinations** from TOTO 3/10 lottery
 - It's like 6/49 but with less combinations



```
for (int i1 = 1; i1 <= 8; i1++) {  
    for (int i2 = i1 + 1; i2 <= 9; i2++) {  
        for (int i3 = i2 + 1; i3 <= 10; i3++) {  
            System.out.printf("%d %d %d\n", i1, i2, i3,);  
        }  
    }  
}
```



Practice: Conditionals and Loops

Live Exercises in Class (Lab)



Bitwise Operations

Playing with bits

Bits

- The smallest **data units** in the computer (either **0** or **1**)
- **Numbers** consist of bits

16-bit integer

Byte is a **8-bit** integer

```
byte a = 3;           // 00000011
short b = 3;          // 00000000 00000011
int c = 5;             // ... 00000000 00000101
```

32-bit integer

- How to print a **binary** number to the console?

```
System.out.println(Integer.toBinaryString(result));
```


Operations with Bits

- Used on numbers (**byte, short, int, long**)
- Applied **bit by bit**
- Bits can be **shifted** using the **<<** and **>>** operators

```
int n = 520;           // 00000010 00001000
int maskRight = n >> 2; // 00000000 10000010
int bitLeft = n << 3;   // 00010000 01000000

System.out.println(bit); // 82
System.out.println(bit); // 4160
```

NOT and OR operators

- **NOT-operator** \sim turns all **0** to **1** and all **1** to **0**

Unary operation

a	1	0
$\sim a$	0	1

Like **!** for Boolean expressions

- **OR-operator** $|$ returns **0**, only if both operands are **0**

Binary
operation

a	0	1	0	1
b	0	0	1	1
$a b$	0	1	1	1

Like **||** for Boolean expressions

AND and XOR operators

- **AND-operator** `&` returns **1**, only if both operands are **1**

a	0	1	0	1
b	0	0	1	1
a & b	0	0	0	1

Like `&&` for Boolean expressions

- **XOR-operator** `^` returns **1**, only if both operands are **different**

Like `^` for Boolean expressions

a	0	1	0	1
b	0	0	1	1
a ^ b	0	1	1	0

Problem: Extract Bit from Integer

- Extract the value of given **bit at index p** by a given:
 - Integer **n**
- Note that the bits are counted from **right to left, starting from bit 0.**

5 2



1

5 = 0101

1 = 0001

Check your solution here: <https://judge.softuni.bg/Contests/382>

Solution: Extract Bit from Integer

- How to get the bit at position **p** from a number **n**?

```
int n = 291;           // 00000001 00100011
int p = 5;

int mask = n >> p;     // 00000000 00001001
int bit = mask & 1;    // 00000000 00000001

System.out.println(bit); // 1
```



Check your solution here: <https://judge.softuni.bg/Contests/382>

Problem: Modify a Bit

- You are given:
 - Integer **n**
 - Position **p**
 - Bit value **v (0 or 1)**
- Modifies **n** to hold the value **v** at the position **p** while **preserving all other bits in n**.

$$9 = 1001$$

$$0 = 00000$$

9 0 1



512

Check your solution here: <https://judge.softuni.bg/Contests/382>

Solution: Modify a Bit

- How to set the bit at position **p** to **0**?

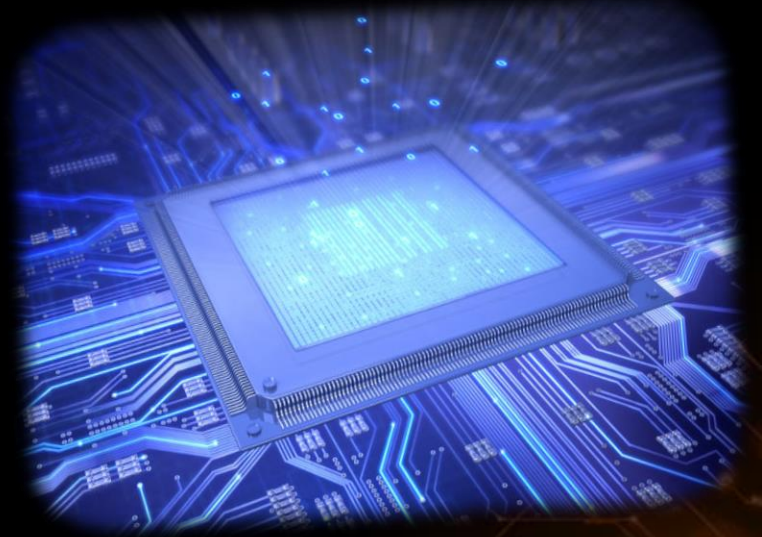
```
int n = 291;           // 00000001 00100011
int p = 5;
int mask = ~(1 << p);  // 11111111 11011111
int result = n & mask;  // 00000001 00000011
System.out.println(result); // 259
```

- How to set the bit at position **p** to **1**?

```
int n = 291;           // 00000001 00100011
int p = 4;
int mask = 1 << p;      // 00000000 00010000
int result = n | mask;   // 00000001 00110011
System.out.println(result); // 307
```



Check your solution here: <https://judge.softuni.bg/Contests/382>



Practice: Bitwise Operations

Live Exercises in Class (Lab)

Summary

- Java supports limited set of primitive data types
- **Scanner** and **System.out.printf()** provide formatted input / output
- Java supports the classical **if-else** and **switch-case**
- Java supports the classical **loop constructions**
- Java provides **bitwise** and **bit shift operations** on integral types



Intro to Java



Questions?



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
- Software University Foundation
 - <http://softuni.foundation/>
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



**Software
University**



**SoftUni
Foundation**

